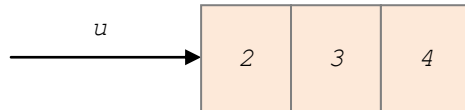


# Tentamen: lösning

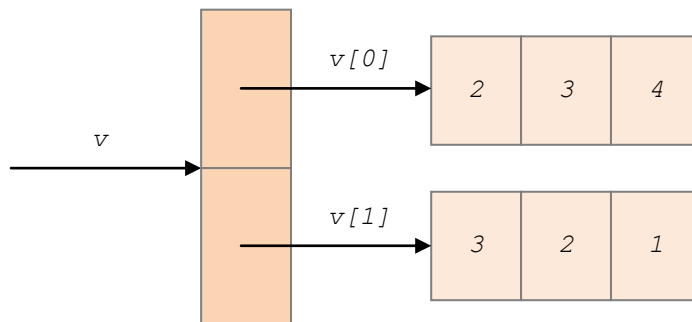
## Uppgifter: lösningar

### Uppgift 1 (3 poäng + 3 poäng)

a) (3 poäng)



b) (3 poäng)



### Uppgift 2 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

```
public static Triangle maxTriangle (Triangle[] triangles)
{
    if (triangles.length == 0)
        throw new java.lang.IllegalArgumentException ("empty array");

    Triangle    maxTr = triangles[0];
    double      maxPerimeter = maxTr.perimeter ();
    for (int pos = 1; pos < triangles.length; pos++)
        if (triangles[pos].perimeter () > maxPerimeter)
        {
            maxTr = triangles[pos];
            maxPerimeter = maxTr.perimeter ();
        }

    return maxTr;
}
```

b) (3 poäng)

```
public static double totalArea (Triangle[] triangles)
{
    double      totalAr = 0;
    for (Triangle t : triangles)
```

```

        totalAr = totalAr + t.area ();

    return totalAr;
}

```

c) (3 poäng)

```

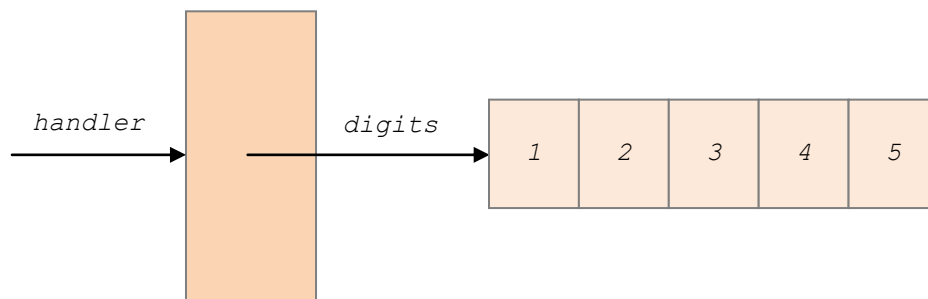
Triangle[]    triangles = { new Triangle (3, 4, 5),
                             new Triangle (4, 4, 4),
                             new Triangle (4, 3, 4),
                             new Triangle (4, 5, 6) };

Triangle      maxTr = maxTriangle (triangles);
double        totalAr = totalArea (triangles);

```

### Uppgift 3 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)



b) (3 poäng)

```

public void circularShiftLeft ()
{
    if (digits.length > 0)
    {
        byte    b = digits[0];
        for (int pos = 0; pos < digits.length - 1; pos++)
            digits[pos] = digits[pos + 1];
        digits[digits.length - 1] = b;
    }
}

```

c) (3 poäng)

```

public void circularShiftRight ()
{
    if (digits.length > 0)
    {
        byte    b = digits[digits.length - 1];
        for (int pos = digits.length - 1; pos > 0; pos--)
            digits[pos] = digits[pos - 1];
        digits[0] = b;
    }
}

```

### Uppgift 4 (3 poäng + 3 poäng + 3 poäng)

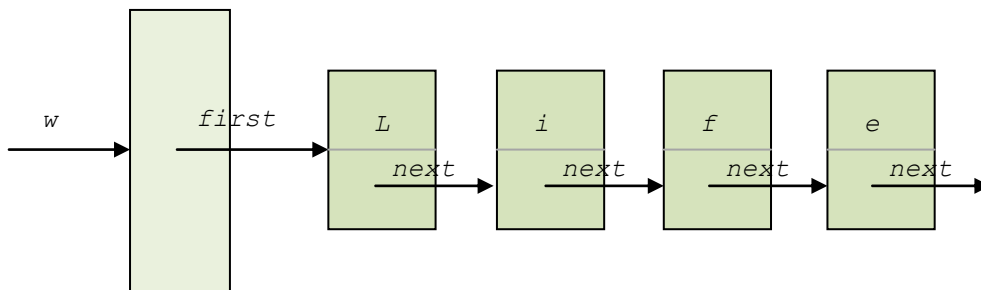
a) (3 poäng)

```

public Word (char[] v)
{
    if (v.length != 0)
    {
        first = new Node (v[0]);
        Node n = first;
        for (int pos = 1; pos < v.length; pos++)
        {
            n.next = new Node (v[pos]);
            n = n.next;
        }
    }
}

```

b) (3 poäng)



c) (3 poäng)

**[LightLove]**

## Uppgift 5 (3 poäng + 3 poäng + 3 poäng)

a) (3 poäng)

**[5, 4, 3, 2, 1]****[1, 5, 4, 3, 2]****[1, 2, 5, 4, 3]****[1, 2, 3, 5, 4]****[1, 2, 3, 4, 5]****[1, 2, 3, 4, 5]**

b) (3 poäng)

Förvillkor:

En sekvens, vars element kan jämföras med operatören mindre (&lt;), är given:

 $x_1, x_2, x_3, \dots, x_n$ , där  $n$  är ett positivt heltal

Eftervillkor:

Sekvensen är sorterad i en stigande ordning:

 $x_{i1} < x_{i2} < x_{i3} < \dots < x_{in}$ 

Steg i algoritmen:

```

sort (n, x[1], x[2], ..., x[n])
{

```

```
for pos = 1, 2, ..., n - 1
{
    for p = pos + 1, pos + 2, ..., n
        if (x[p] < x[pos])
            exchange x[p] and x[pos]
    }
}
```

c) (3 poäng)

För att fylla i den första positionen med rätt element, utförs  $n - 1$  elementjämförelser. För att fylla i den andra positionen med rätt element utförs  $n - 2$  jämförelser. Antalet jämförelser minskar med 1 för varje ny position. Det totala antalet jämförelser är:

$$(n - 1) + (n - 2) + \dots + 1 = n(n - 1) / 2$$

Motsvarande komplexitetsfunktion är:

$$t(n) = n(n - 1) / 2$$

$$t(n) = n^2 / 2 - n / 2$$

För stora  $n$  dominerar termen med  $n^2$ . Därför:

$$t(n) \in \theta(n^2)$$

Algoritmen är kvadratisk när det gäller antalet elementjämförelser.

När det gäller antalet elementutbyten, uppstår värsta fall för algoritmen när sekvensen i början är sorterad i omvänd ordning. I så fall utförs ett elementutbyte varje gång två element jämförs. Det betyder att det finns så många elementutbyten som jämförelser, alltså:

$$w(n) = n(n - 1) / 2$$

$$w(n) \in \theta(n^2)$$